

On problems equivalent to $(\min, +)$ -convolution*

Marek Cygan[†] Marcin Mucha[†] Karol Węgrzycki[†] Michał Włodarczyk[†]

February 27, 2017

Abstract

In the recent years, significant progress has been made in explaining apparent hardness of improving over naive solutions for many fundamental polynomially solvable problems. This came in the form of conditional lower bounds – reductions to one of problems assumed to be hard. These include 3SUM, All-Pairs Shortest Paths, SAT and Orthogonal Vectors, and others.

In the $(\min, +)$ -convolution problem, the goal is to compute a sequence $(c[i])_{i=0}^{n-1}$, where $c[k] = \min_{i=0, \dots, k} \{a[i] + b[k - i]\}$, given sequences $(a[i])_{i=0}^{n-1}$ and $(b[i])_{i=0}^{n-1}$. This can easily be done in $\mathcal{O}(n^2)$ time, but no $\mathcal{O}(n^{2-\varepsilon})$ algorithm is known for $\varepsilon > 0$. In this paper we undertake a systematic study of the $(\min, +)$ -convolution problem as a hardness assumption.

As the first step, we establish equivalence of this problem to a group of other problems, including variants of the classic knapsack problem and problems related to subadditive sequences. The $(\min, +)$ -convolution has been used as a building block in algorithms for many problems, notably problems in stringology. It has also already appeared as an ad hoc hardness assumption. We investigate some of these connections and provide new reductions and other results.

*This work is part of a project TOTAL that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 677651).

[†]Institute of Informatics, University of Warsaw, Poland, {cygan, mucha, k.wegrzycki, m.wlodarczyk}@mimuw.edu.pl

1 Introduction

1.1 Hardness in P

For many problems there exist ingenious algorithms that significantly improve upon the naive approach in terms of time complexity. On the other hand, for some fundamental problems, the naive algorithms are still the best known, or have been improved upon only slightly. To some extent this has been explained by the $P \neq NP$ conjecture. However, for many problems even the naive approaches lead to polynomial algorithms, and the $P \neq NP$ conjecture does not seem to be particularly useful for proving polynomial lower bounds.

In the recent years, significant progress has been made in establishing such bounds, conditioned on conjectures other than $P \neq NP$, each of them claiming time complexity lower bounds for a different problem. And so, conjecture that there is no $\mathcal{O}(n^{2-\epsilon})$ algorithm for 3SUM problem¹ implies hardness for the problems in the computational geometry [23] and dynamic algorithms [36]. The conjecture that All-Pairs Shortest Paths (APSP) is hard implies hardness of finding graph radius, graph median and some dynamic problems (see [42] for survey). Finally, the Strong Exponential Time Hypothesis (SETH) introduced in [27, 28] that has been used extensively to prove hardness of parametrized problems, recently lead to polynomial lower bounds via the intermediate Orthogonal Vectors problem (see [40]). These include bounds for Edit Distance [4], Longest Common Subsequence [10, 2], and other [42].

It is worth noting that in many cases the results mentioned are not only showing the hardness of the problem in question, but also that it is computationally equivalent to the underlying hard problem. This leads to clusters of equivalent problems being formed, each cluster corresponding to a single hardness assumption (see [42, Figure 1]).

As Christos H. Papadimitriou is quoted to say „*There is nothing wrong with trying to prove that $P=NP$ by developing a polynomial-time algorithm for an NP-complete problem. The point is that without an NP-completeness proof we would be trying the same thing without knowing it!*” [35]. In the same spirit, these new conditional hardness results have cleared the polynomial landscape by showing that there really are not that many hard problems.

1.2 Hardness of MinConv

In this paper we propose yet another hardness assumption in the MINCONV problem.

MINCONV

Input: Sequences $(a[i])_{i=0}^{n-1}, (b[i])_{i=0}^{n-1}$

Task: Output sequence $(c[i])_{i=0}^{n-1}$, such that $c[k] = \min_{i+j=k} (a[i] + b[j])$

This problem has been used as a hardness assumption before for at least two specific problems [32, 5], but to the best of our knowledge no attempts have been made to systematically study the neighborhood of this problem in the polynomial complexity landscape. To be more precise, we consider the following.

Conjecture 1. *There is no $\mathcal{O}(n^{2-\epsilon})$ algorithm for MINCONV, for $\epsilon > 0$.*

¹We included all problem definitions together with known results concerning these problems in Section 2. This is to keep the introduction relatively free of technicalities.

Let us first look at the place occupied by MINCONV in the landscape of established hardness conjectures. Figure 1 shows known reductions between these conjectures and includes MINCONV. Bremner et al. [8] showed reduction from MINCONV to APSP. It is also known [5, 1] that MINCONV can be reduced to 3SUM (to the best of our knowledge no such reduction has been published before, and we provide the details in Appendix C). Note that a reduction from 3SUM or APSP to MINCONV would imply a reduction between 3SUM and APSP, which is a major open problem in the area [42]. No relation is known between MINCONV and SETH or OV.

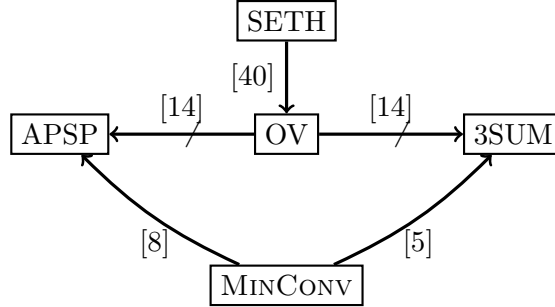


Figure 1: The relationship between popular conjectures. A reduction from OV to 3SUM or APSP contradicts the nondeterministic version of SETH [14, 42] (these arrows are striked-out).

In this paper we study three broad categories of problems. The first category consists of the classic 0/1 KNAPSACK and its variants, which we show to be essentially equivalent to MINCONV. This is perhaps somewhat surprising, given recent progress of Bringmann [9] for SUBSETSUM, which is a special case of 0/1 KNAPSACK. However, note that the Bringmann’s algorithm [9] (as well as in other efficient solutions for SUBSETSUM) is build upon the idea of composing solutions using the (\vee, \wedge) -convolution, which can implemented efficiently using Fast Fourier Transform (FFT). The corresponding composition operation for 0/1 KNAPSACK is MINCONV (see Appendix B for details).

The second category consists problems directly related to MINCONV. This includes decision versions of MINCONV, and problems related to the notion of subadditivity. Any subadditive sequence a with $a[0] = 0$ is an idempotent of MINCONV, so it is perhaps natural that these problems turn out to be equivalent to MINCONV.

Finally, we investigate problems that have previously been shown to be related to MINCONV, and contribute some new reductions, or simplify existing ones.

2 Problem definitions and known results

2.1 3SUM

3SUM

Input: Sets of integers A, B, C , each of size n

Task: Decide whether there exist $a \in A, b \in B, c \in C$ such that $a + b = c$

The 3SUM problem is the first problem that was considered as a hardness assumption in P. It admits a simple $\mathcal{O}(n^2 \log n)$ algorithm but the existence of an $\mathcal{O}(n^{2-\epsilon})$ algorithm remains a big open problem. The first lower bounds based on hardness of 3SUM appeared in 1995 [23] and some other examples can be found in [6, 36, 43]. The current best algorithm for 3SUM runs in

slightly subquadratic expected time $\mathcal{O}\left((n^2/\log^2 n)(\log \log n)^2\right)$ [6]. An $\mathcal{O}(n^{1.5}\text{polylog}(n))$ algorithm is possible on the nondeterministic Turing machine [14]. The 3SUM problem is known to be subquadratically equivalent to its convolution version [36].

3SUMCONV

Input: Sequences a, b, c , each of length n

Task: Decide whether there exist i, j such that $a[i] + b[j] = c[i + j]$

Both problems are sometimes considered with real weights but in this work we restrict only to the integer setting.

2.2 MinConv

We have already defined the MINCONV problem in Subsection 1.2. Note that it is equivalent (just by negating elements) to the analogous MAXCONV problem.

MAXCONV

Input: Sequences $(a[i])_{i=0}^{n-1}, (b[i])_{i=0}^{n-1}$

Task: Output sequence $(c[i])_{i=0}^{n-1}$, such that $c[k] = \max_{i+j=k}(a[i] + b[j])$

We describe our contribution in terms of MINCONV as this version has been already been heavily studied. However, in the theorems and proofs we use MAXCONV, as it is easier to work with. We will also work with a decision version of the problem.

MAXCONV UPPERBOUND

Input: Sequences $(a[i])_{i=0}^{n-1}, (b[i])_{i=0}^{n-1}, (c[i])_{i=0}^{n-1}$

Task: Decide whether $c[k] \geq \max_{i+j=k}(a[i] + b[j])$ for all k

If we replace the latter condition with $c[k] \leq \max_{i+j=k}(a[i] + b[j])$ we obtain a similar problem MAXCONV LOWERBOUND. Yet another statement of a decision version asks whether a given sequence is a self upper bound with respect to MAXCONV, i.e., if it is superadditive. From the perspective of MINCONV we may ask an analogous question about being subadditive (again equivalent by negating elements). As far as we know, the computational complexity of these problems has not been studied yet.

SUPERADDITIVITY TESTING

Input: A sequence $(a[i])_{i=0}^{n-1}$

Task: Decide whether $a[k] \geq \max_{i+j=k}(a[i] + a[j])$ for all k

In the standard $(+, \cdot)$ ring, convolution can be computed in $\mathcal{O}(n \log n)$ time by the FFT. A natural line of attacking MINCONV would be to design an analogue of FFT in the $(\min, +)$ -semiring, also called a *tropical semiring*². However, due to the lack of inverse for the min-operation it is unclear if such a transform exists for general sequences. When restricted to convex sequences, one can use a tropical analogue of FFT, namely the Legendre-Fenchel transform [20], which can be performed in linear time [33].

There has been a long line of research dedicated to improve naive algorithm for MINCONV. Bremner et al. [8] gave an $\mathcal{O}(n^2/\log n)$ algorithm for MINCONV, and gave a reduction from MIN-

²In this setting MINCONV is often called $(\min, +)$ -convolution, inf-convolution or epigraphic sum.

CONV to APSP [8, Theorem 13]. Williams [41] gave an $\mathcal{O}(n^3/2^{\Omega(\log n)^{1/2}})$ algorithm for APSP, which implies the best known $\mathcal{O}(n^2/2^{\Omega(\log n)^{1/2}})$ algorithm for MINCONV [16].

Truly subquadratic algorithms for MINCONV exist for monotone increasing sequences with integer values bounded by $\mathcal{O}(n)$. Chan and Lewenstein [16] presented an $\mathcal{O}(n^{1.859})$ randomized algorithm and an $\mathcal{O}(n^{1.864})$ deterministic algorithm for that case. They exploited ideas from additive combinatorics. Bussieck et al. [13] showed that for the random input, MINCONV can be computed in $\mathcal{O}(n \log n)$ expected and $\Theta(n^2)$ worst case time.

If we are satisfied with computing c with a relative error $(1+\epsilon)$ then general MINCONV admits a nearly-linear algorithm [5, 44]. It could be called an FPTAS (fully polynomial-time approximation schema) with a remark that usually this name is reserved for single-output problems for which decision versions are NP-hard.

Using techniques of Carmosino et al. [14] and reduction from MAXCONV UPPERBOUND to 3SUM (see Appendix C) one can construct an $\mathcal{O}(n^{1.5} \text{polylog}(n))$ algorithm working on nondeterministic Turing machines for MINCONV. What is interesting, this running time matches the $\mathcal{O}(n^{1.5})$ algorithm in the nonuniform decision tree model given by Bremner et al. [8]. This result is based on the techniques of Fredman [22, 21]. It remains unclear how to transfer these results to the word-RAM model [8].

2.3 Knapsack

0/1 KNAPSACK

Input: A set of items \mathcal{I} with given weights and values $((w_i, v_i))_{i \in \mathcal{I}}$, capacity t

Task: Find the maximal total value of the items subset $\mathcal{I}' \subseteq \mathcal{I}$ such that $\sum_{i \in \mathcal{I}'} w_i \leq t$

If we are allowed to take multiple copies of a single item then we obtain the UNBOUNDED KNAPSACK problem. The decision versions of both problems are known to be NP-hard [24] but there are classical algorithms based on dynamic programming with a pseudo-polynomial running time $\mathcal{O}(nt)$ [7]. In fact they solve more general problems, i.e., 0/1 KNAPSACK⁺ and UNBOUNDED KNAPSACK⁺, where we are asked to output answers for each $0 < t' \leq t$. There is also a long line of research on FPTAS for KNAPSACK with the current best running times respectively $\mathcal{O}(n \log \frac{1}{\epsilon} + \frac{1}{\epsilon^3} \log^2 \frac{1}{\epsilon})$ for 0/1 KNAPSACK [30] and $\mathcal{O}(n + \frac{1}{\epsilon^2} \log^3 \frac{1}{\epsilon})$ for UNBOUNDED KNAPSACK [29].

2.4 Other problems related to MinConv

TREE SPARSITY

Input: A rooted tree T with a weight function $x : V(T) \rightarrow \mathbb{N}_{\geq 0}$, parameter k

Task: Find the maximal total weight of rooted subtree of size k

The TREE SPARSITY problem admits an $\mathcal{O}(nk)$ algorithm, which was at first invented for restricted case of balanced trees [15] and generalised later [5]. There is also a nearly-linear FPTAS based on the FPTAS for MINCONV [5]. It is known that an $\mathcal{O}(n^{2-\epsilon})$ algorithm for TREE SPARSITY entails a subquadratic algorithm for MINCONV [5].

MCSP

Input: A sequence $(a[i])_{i=0}^{n-1}$

Task: Output the maximal sum of k consecutive elements for each k

There is a trivial $\mathcal{O}(n^2)$ algorithm for MCSP and a nearly-linear FPTAS based on the FPTAS for MINCONV [17]. To the best of our knowledge, this is the first problem to have been explicitly proven to be subquadratically equivalent with MINCONV [32]. Our reduction to SUPERADDITIVITY TESTING allows us to significantly simplify the proof (see Section 6.1).

l_p -NECKLACE ALIGNMENT

Input: Sequences $(x[i])_{i=0}^{n-1}, (y[i])_{i=0}^{n-1}$ describing locations of beads on a circle

Task: Output the cost of the best alignment in p -norm, i.e., $\sum_{i=0}^{n-1} d(x[i] + c, y[\pi(i)])^p$ where c is a circular shift, π is a permutation, and d is a distance function on a circle

For $p = \infty$ we are interested in bounding the maximal distance between any two matched beads. The problem initially emerged for $p = 1$ during the research on geometry of musical rhythm [38]. The family of NECKLACE ALIGNMENT problems has been systematically studied by Bremner et al. [8] for various values of p , in particular $1, 2, \infty$. For $p = 2$ they presented an $\mathcal{O}(n \log n)$ algorithm based on Fast Fourier Transform. For $p = \infty$ the problem was reduced to MINCONV which led to a slightly subquadratic algorithm.

Although it is more natural to state the problem with inputs from $[0, 1)$, we find it more convenient to work with integer sequences that describe a necklace after scaling.

Fast $o(n^2)$ algorithms for MINCONV have also found applications in text algorithms. Moosa and Rahman [34] reduced the *Indexed Permutation Matching* to MINCONV and obtained $o(n^2)$ algorithm. Burcsi et al. [11] used MINCONV to get faster algorithms for *Jumbled Pattern Matching* and described how finding dominating pairs can be used to solve MINCONV. Later Burcsi et al. [12] showed that fast MINCONV can also be used to get faster algorithms for a decision version of the *Approximate Jumbled Pattern Matching* over binary alphabets.

3 New results summary

Figure 2 illustrates the technical contributions of this paper. The long ring of reductions on the left side of the figure is summarized below.

Theorem 2. *The following statements are equivalent:*

1. *There exists an $\mathcal{O}(n^{2-\varepsilon})$ algorithm for MAXCONV for some $\varepsilon > 0$.*
2. *There exists an $\mathcal{O}(n^{2-\varepsilon})$ algorithm for MAXCONV UPPERBOUND for some $\varepsilon > 0$.*
3. *There exists an $\mathcal{O}(n^{2-\varepsilon})$ algorithm for SUPERADDITIVITY TESTING for some $\varepsilon > 0$.*
4. *There exists an $\mathcal{O}((n+t)^{2-\varepsilon})$ algorithm for UNBOUNDED KNAPSACK for some $\varepsilon > 0$.*
5. *There exists an $\mathcal{O}((n+t)^{2-\varepsilon})$ algorithm for 0/1 KNAPSACK for some $\varepsilon > 0$.*

Theorem 2 is split into five implications, presented separately as Theorems 3,4,5,6 and 7 in Section 5. While Theorem 2 has a relatively short and simple statement, it is not the strongest possible version of the equivalence. In particular, one can show analogous implications for subpolynomial improvements, such as the $\mathcal{O}(n^2/2^{\Omega(\log n)^{1/2}})$ algorithm of Williams [41]. The theorems listed above contain stronger versions of the implications.

Section 6 is devoted to the remaining arrows in Figure 2. In Subsection 6.1, we show that using Theorem 2 we can obtain an alternative proof of the equivalence of MCSP and MAXCONV (and so

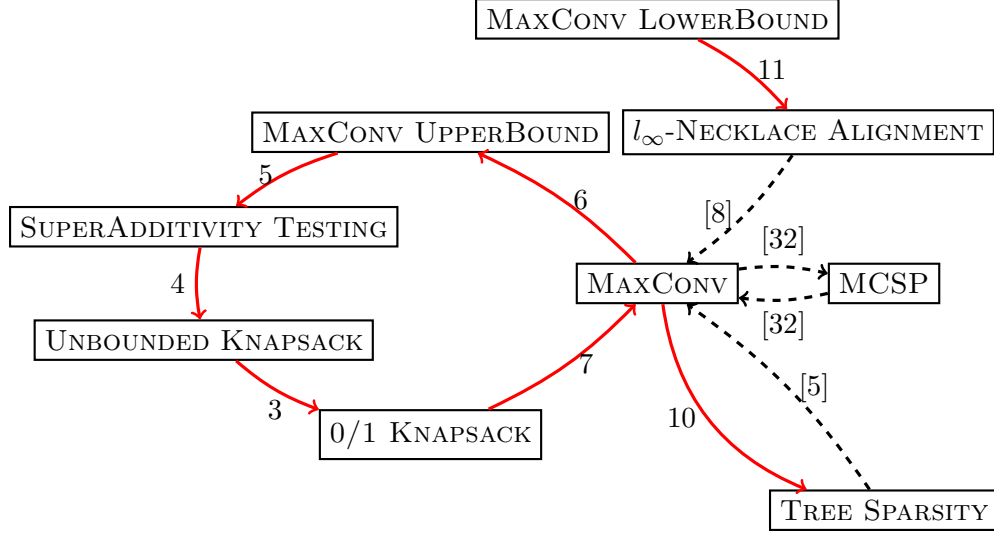


Figure 2: Summary of reductions in the MINCONV complexity class. An arrow from problem A to B denotes a reduction from A to B . Black dashed arrows were previously known, red arrows are new results. Numbers next to red arrows point to the corresponding theorems.

also MINCONV), much simpler than the one presented in [32]. In Subsection 6.2, we show that TREE SPARSITY reduces to MAXCONV, complementing the opposite reduction showed in [5]. Finally in Subsection 6.3 we provide some observations on the possible equivalence between l_∞ -NECKLACE ALIGNMENT and MAXCONV.

4 Preliminaries

We present a series of results of the following form: if a problem \mathcal{A} admits an algorithm with running time $T(n)$, then a problem \mathcal{B} admits an algorithm with running time $T'(n)$, where function T' depends on T and n is the length of the input. Our main interest is in showing that $T(n) = \mathcal{O}(n^{2-\epsilon}) \Rightarrow T'(n) = \mathcal{O}(n^{2-\epsilon'})$. Some problems, in particular KNAPSACK, have no simple parameterization and we allow function T to take multiple arguments.

We assume that for all studied problems the input consists of a list of integers within $[-W, W]$. For the sake of readability we omit W as a running time parameter and we allow function T to hide $\text{polylog}(W)$ factors. As sometimes the size of the input grows in the reduction, we restrict ourselves to a class of functions satisfying $T(cn) = \mathcal{O}(T(n))$ for a constant c . This is justified as we mainly focus on functions of the form $T(n) = n^\alpha$. In some reductions the integers in the new instance may increase to $\mathcal{O}(nW)$. In that case we multiply the running time by $\log n$ to take into account the overhead of performing arithmetic operations. All logarithms are base 2.

5 Main reductions

Theorem 3 (UNBOUNDED KNAPSACK \rightarrow 0/1 KNAPSACK). *A $T(n, t)$ algorithm for 0/1 KNAPSACK implies an $\mathcal{O}(T(n, t) \log t)$ algorithm for UNBOUNDED KNAPSACK.*

Proof. Consider an instance of UNBOUNDED KNAPSACK with the capacity t and the set of items given as weight-value pairs $((w_i, v_i))_{i \in \mathcal{I}}$. Construct an equivalent 0/1 KNAPSACK instance with the same t and the set of items $((2^j w_i, 2^j v_i))_{i \in \mathcal{I}, 0 \leq j \leq \log t}$. Let $X = (x_i)_{i \in \mathcal{I}}$ be the list of multiplicities of items chosen in a solution to the UNBOUNDED KNAPSACK problem. Of course, $x_i \leq t$. Define $(x_i^j)_{0 \leq j \leq \log t}$, $x_i^j \in \{0, 1\}$ to be the binary representation of x_i . Then the vector $(x_i^j)_{i \in \mathcal{I}, 0 \leq j \leq \log t}$ induces a solution to 0/1 KNAPSACK with the same total weight and value. The described mapping can be reverted what implies the equivalence between the instances and proves the claim. \square

Theorem 4 (SUPERADDITIVITY TESTING \rightarrow UNBOUNDED KNAPSACK). *If UNBOUNDED KNAPSACK can be solved in time $T(n, t)$ then SUPERADDITIVITY TESTING admits an algorithm with running time $\mathcal{O}(T(n, n) \log n)$.*

Proof. Let $(a[i])_{i=0}^{n-1}$ be a non-negative monotonic sequence.³ Set $D = \sum_{i=0}^{n-1} a[i] + 1$ and construct an UNBOUNDED KNAPSACK instance with the set of items $((i, a[i]))_{i=0}^{n-1} \cup ((2n-1-i, D-a[i]))_{i=0}^{n-1}$ and $t = 2n-1$. It is always possible to gain D by taking two items $(i, a[i]), (2n-1-i, D-a[i])$ for any i . We will claim that the answer to the constructed instance equals D if and only if a is superadditive.

If a is not superadditive, then there are i, j such that $a[i] + a[j] > a[i+j]$. Choosing

$$((i, a[i]), (j, a[j]), (2n-1-i-j, D-a[i+j]))$$

gives a solution of value exceeding D .

Now assume that a is superadditive. Observe that any feasible knapsack solution may contain at most one item with weight exceeding $n-1$. On the other hand, the optimal solution has to include one such item because the total value of the lighter ones is less than D . Therefore the optimal solution contains an item $(2n-1-k, D-a[k])$ for some $k < n$. The total weight of the rest of the solution is at most k . As a is superadditive, we can replace any pair $(i, a[i]), (j, a[j])$ with the item $(i+j, a[i+j])$ without decreasing the value of the solution. By repeating this argument, we end up with a single item lighter than n . The sequence a is monotonic so it is always profitable to replace this item with a heavier one, as long as the load does not exceed t . We conclude that the optimal solution must be of form $((k, a[k]), (2n-1-k, D-a[k]))$, which finishes the proof. \square

Theorem 5 (MAXCONV UPPERBOUND \rightarrow SUPERADDITIVITY TESTING). *If SUPERADDITIVITY TESTING can be solved in time $T(n)$ then MAXCONV UPPERBOUND admits an algorithm with running time $\mathcal{O}(T(n) \log n)$.*

Proof. We start with reducing the instance of MAXCONV UPPERBOUND to the case of non-negative monotonic sequences. Observe that condition $a[i] + b[j] \leq c[i+j]$ can be rewritten as $(C + a[i] + Di) + (C + b[j] + Dj) \leq 2C + c[i+j] + D(i+j)$ for any constants C, D . Hence, replacing sequences $(a[i])_{i=0}^{n-1}, (b[i])_{i=0}^{n-1}, (c[i])_{i=0}^{n-1}$ with $a'[i] = C + a[i] + Di, b'[i] = C + b[i] + Di, c'[i] = 2C + c[i] + Di$ leads to an equivalent instance. We can thus pick C, D of magnitude $\mathcal{O}(W)$ to ensure that all elements are non-negative and do not exceed the successor. The values in the new sequences may rise up to $\mathcal{O}(nW)$.

From now we can assume the given sequences to be non-negative and monotonic. Define K to be the maximal value occurring in any sequence. Construct a sequence e of length $4n$ as follows. For $i \in [0, n-1]$ set $e[i] = 0, e[n+i] = K + a[i], e[2n+i] = 4K + b[i], e[3n+i] = 5K + c[i]$. If

³For a technical reduction of SUPERADDITIVITY TESTING to this case see Appendix A

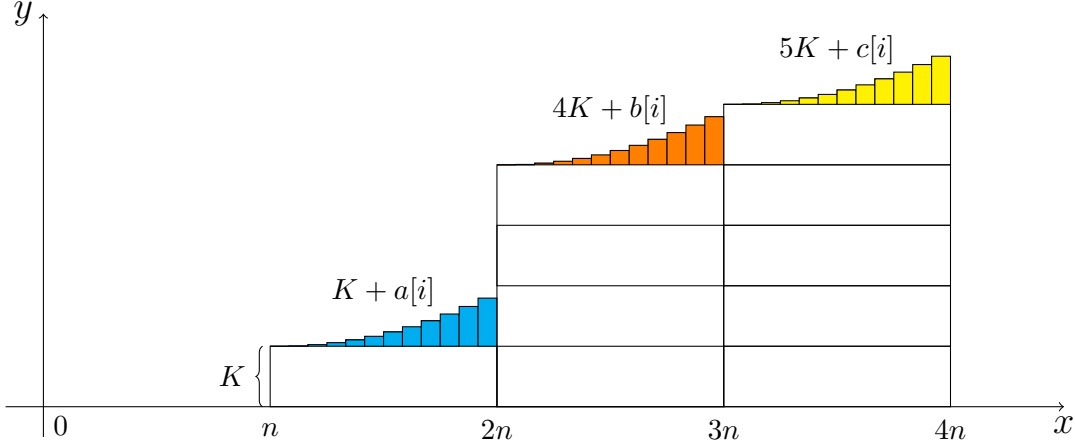


Figure 3: Graphical interpretation of the sequence e in Theorem 5. The height of boxes equals K .

there is $a[i] + b[j] > c[i + j]$ for some i, j , then $e[n + i] + e[2n + j] > e[3n + i + j]$ and therefore e is not superadditive. We now show that otherwise e must be superadditive.

Assume w.l.o.g. $i \leq j$. The case $i < n$ can be ruled out because it implies $e[i] = 0$ and $e[i] + e[j] \leq e[i + j]$ for any j as e is monotonic. If $i \geq 2n$, then $i + j \geq 4n$, so we can restrict to $i \in [n, 2n - 1]$. We can also clearly assume $j < 3n$. If $j \in [n, 2n - 1]$, then $e[i] + e[j] \leq 4K \leq e[i + j]$. Finally, $j \in [2n, 3n - 1]$ corresponds to the original condition. \square

Theorem 6 (MAXCONV \rightarrow MAXCONV UPPERBOUND). *A $T(n)$ algorithm for MAXCONV UPPERBOUND implies an $\mathcal{O}(T(\sqrt{n})n \log n)$ algorithm for MAXCONV.*

The proof of the reduction from MAXCONV to MAXCONV UPPERBOUND has been independently given recently in [5]. For completeness we give our proof in Appendix D.

Theorem 7 (0/1 KNAPSACK \rightarrow MAXCONV). *A $T(n)$ algorithm for MAXCONV implies an $\mathcal{O}(T(t \log t) \log^3(n/\delta) \log n)$ for 0/1 KNAPSACK that outputs the correct answer with probability at least $1 - \delta$.*

Corollary 7.1. *An $\mathcal{O}((n + t)^{2-\epsilon})$ time algorithm for 0/1 KNAPSACK implies an $\mathcal{O}(t^{2-\epsilon'} + n)$ time algorithm for 0/1 KNAPSACK⁺.*

The proof follows the approach of Bringmann [9], and we present it in Appendix B.

6 Other problems related to MinConv

6.1 Maximum consecutive subsums problem

The MAXIMUM CONSECUTIVE SUBSUMS PROBLEM (MCSP) is to the best of our knowledge the first problem that has been explicitly proven to be subquadratically equivalent with MINCONV [32]. The reduction from MCSP to MAXCONV is only shown for completeness, but the reduction in the opposite direction is much simpler than the original one.

Theorem 8 (MCSP \rightarrow MAXCONV). *If MAXCONV can be solved in time $T(n)$ then MCSP admits an algorithm with running time $\mathcal{O}(T(n))$.*

Proof. Let $(a[i])_{i=0}^{n-1}$ be the input sequence. Construct sequences of length $2n$ as follows: $b[k] = \sum_{i=0}^k a[i]$ for $k < n$, $c[k] = -\sum_{i=0}^{n-k-1} a[i]$ for $k \leq n$ (empty sum equals 0) and otherwise $b[k] = c[k] = -D$, where D is two times larger than any partial sum. Observe that

$$(b \oplus^{\max} c)[n+k-1] = \max_{\substack{0 \leq j < n \\ 0 \leq n+k-j-1 \leq n}} \sum_{i=0}^j a[i] - \sum_{i=0}^{j-k} a[i] = \max_{k-1 \leq j < n} \sum_{i=j-k+1}^j a[i], \quad (1)$$

so we can read the maximum consecutive sum for each length k after performing MAXCONV. \square

Theorem 9 (SUPERADDITIVITY TESTING \rightarrow MCSP). *If MCSP can be solved in time $T(n)$ then SUPERADDITIVITY TESTING admits an algorithm with running time $\mathcal{O}(T(n))$.*

Proof. Let $(a[i])_{i=0}^{n-1}$ be the input sequence and $b[i] = a[i+1] - a[i]$. The superadditivity condition $a[k] \leq a[k+j] - a[j]$ (for all possible k, j) can be translated into $a[k] \leq \min_{0 \leq j < n-k} \sum_{i=j}^{k+j-1} b[i]$ (for all k), so computing MCSP vector on $(-b[i])_{i=0}^{n-2}$ suffices to check if the above condition holds. \square

6.2 Tree Sparsity

Theorem 10 (TREE SPARSITY \rightarrow MAXCONV). *If MAXCONV can be solved in time $T(n)$ and the function T is superadditive then TREE SPARSITY admits an algorithm with running time $\mathcal{O}(T(n) \log^2 n)$.*

Proof. We take advantage of the heavy-light decomposition introduced by Sleator and Tarjan [37]. This technique has been utilized by Backurs et al. [5] in order to transform a nearly-linear PTAS for MAXCONV to a nearly-linear PTAS for TREE SPARSITY. The reduction for exact subquadratic algorithms is different in the second phase though.

We construct a *spine* with a *head* s_1 at the root of the tree. We define s_{i+1} to be the child of s_i with the larger subtree (in case of draw we choose any child) and the last node in the spine is a leaf. The other children of nodes s_i become heads for analogous spines so the whole tree gets covered. Note that every path from a leaf to the root intersects at most $\log n$ spines because each spine transition doubles the subtree size.

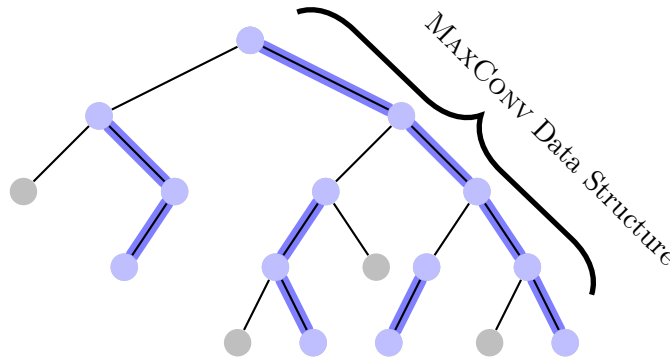


Figure 4: Schema of spine decomposition [5]. Blue edges represent edges on the spine. For each spine we build efficient data structure that uses MAXCONV (curly brackets). There are at most $\mathcal{O}(\log n)$ different spines on path from a leaf to the root.

For a node v with a subtree of size m we define the sparsity vector $(x^v[0], x^v[1], \dots, x^v[m])$ with the weights of the heaviest subtrees rooted at v with fixed sizes. We are going to compute sparsity vectors for all heads of spines in the tree recursively. Let $(s_i)_{i=1}^\ell$ be a spine with a head v and let u^i indicate the sparsity vector for the child of s_i being a head (i.e., the child with the smaller subtree). If s_i has less than two children we treat u^i as a vector (0) .

For an interval $[a, b] \subseteq [1, \ell]$ let $u^{a,b} = u^a \oplus^{\max} u^{a+1} \oplus^{\max} \dots \oplus^{\max} u^b$ and $y^{a,b}[k]$ be the maximum weight of a subtree of size k rooted at s_a and not containing s_{b+1} . Let $c = \lfloor \frac{a+b}{2} \rfloor$. The \oplus^{\max} operator is associative so $u^{a,b} = u^{a,c} \oplus^{\max} u^{c+1,b}$. To compute the second vector we consider two cases: whether the optimal subtree contains s_{c+1} or not.

$$\begin{aligned} y^{a,b}[k] &= \max \left[y^{a,c}[k], \sum_{i=a}^c x(s_i) + \max_{k_1+k_2=k-(c-a+1)} (u^{a,c}[k_1] + y^{c+1,b}[k_2]) \right] \\ &= \max \left[y^{a,c}[k], \sum_{i=a}^c x(s_i) + (u^{a,c} \oplus^{\max} y^{c+1,b})[k - (c - a + 1)] \right] \end{aligned}$$

Using the presented formulas we reduce the problem of computing $x^v = y^{1,\ell}$ to subproblems for intervals $[1, \frac{\ell}{2}]$ and $[\frac{\ell}{2} + 1, \ell]$ and results are merged with two $(\max, +)$ -convolutions. Proceeding further we obtain $\log \ell$ levels of recursion, where the sum of convolution sizes on each level is $\mathcal{O}(m)$, what results in the total running time $\mathcal{O}(T(m) \log m)$ (recall that T is superadditive with respect to the first argument).

The second type of recursion comes from the spine decomposition. There are at most $\log n$ levels of recursion with the cumulative sum of subtrees bounded by n on each level, what proves the claim. \square

6.3 l_∞ -Necklace Alignment

In this section we study the l_∞ -NECKLACE ALIGNMENT alignment problem that was proved to reduce to MINCONV [8]. We are unable to reduce any of the problems equivalent to MINCONV to this problem, but we do reduce a related problem - MAXCONV LOWERBOUND. We also elaborate on why obtaining a full reduction is difficult.

Theorem 11 (MAXCONV LOWERBOUND \rightarrow l_∞ -NECKLACE ALIGNMENT). *If l_∞ -NECKLACE ALIGNMENT can be solved in time $T(n)$ then MAXCONV LOWERBOUND admits an algorithm with running time $\mathcal{O}(T(n) \log n)$.*

Proof. Let a, b, c be the input sequences to MAXCONV LOWERBOUND. We call a sum of form $e_1[k_1] + e_2[k_2] + \dots + e_m[k_m]$, where $e_i \in \{a, b, c\}$, a *combination*, and we define its order as $\sum_{i=1}^m k_i$. If an element $e_i[k_i]$ occurs with minus, we subtract k_i .

We can assume the following properties of the input sequences w.l.o.g.

1. *We may assume the sequences are non-negative and $a[i] \leq c[i]$ for all i .* Just add C_1 to a , $C_1 + C_2$ to b , and $2C_1 + C_2$ to c for appropriate positive constants C_1, C_2 .
2. *We can artificially append an element $b[n]$ larger than value of any combination of order n and length bounded by a constant L .* Alternatively, we can say that combinations of order 0 with a positive coefficient at $b[n]$ have positive value. Initially, we enforce this property by setting $b[n]$ as the maximum absolute value of an element times L .

3. Any combination of positive order and length bounded by L has a non-negative value. Add a linear function Di to all sequences. As the order of combination is positive, the factors at D sum up to a positive value. It suffices to choose D equal to the maximum absolute value of an element times L . Note that previous inequalities compare combinations of the same order so they stay unaffected.

The values of the elements might increase to $\mathcal{O}(nWL^2)$. For the rest of this proof we will use $L = 10$. Let $B = b[n]$, $B_1 = b[n-1]$, $B_2 = b[n] - b[1]$. We define necklaces x, y of length $2B$ with $2n$ beads each. The property (3) implies monotonicity of the sequences so the beads are given in the right order. We allow two beads to lie at the same place (in particular the first one and the last one in y).

$$\begin{aligned} x &= (a[0], a[1], \dots, a[n-1], B+c[0], B+c[1], \dots, B+c[n-2], B+c[n-1]), \\ y &= (B_1-b[n-1], B_1-b[n-2], \dots, B_1-b[0], B+B_2-b[n-1], B+B_2-b[n-2], \dots, B+B_2-b[1], 2B). \end{aligned}$$

Bremner et al. [8] pointed out that the optimal solution for l_∞ -NECKLACE ALIGNMENT must be non-crossing, so we can consider only matchings of form $(x[i], y[j])$ where $j = i + k \bmod 2n$ and k is fixed. Let $d(x[i], y[j])$ be the forward distance between $x[i]$ and $y[j]$, i.e., $y[j] - x[i]$ plus the length of the necklaces if $j < i$. Define M_k to be $\max_{i \in [0, N)} d(x[i], y[k+i \bmod 2n]) - \min_{i \in [0, N)} d(x[i], y[k+i \bmod 2n])$. In this setting [8, Fact 5] says that for a fixed k the optimal shift provides solution of value $\frac{M_k}{2}$.

We want to show that for $k \in [0, n)$ it holds

$$\begin{aligned} \min_{i \in [0, 2n)} d(x[i], y[k+i \bmod 2n]) &= B_1 - \max_{i+j=n-k-1} (a[i] + b[j]), \\ \max_{i \in [0, 2n)} d(x[i], y[k+i \bmod 2n]) &= B - c[n-k-1]. \end{aligned}$$

There are five types of connections between beads.

$$d(x[i], y[k+i \bmod 2n]) = \begin{cases} B_1 - a[i] - b[n-k-1-i] & i \in [0, n-k-1], & \text{(I)} \\ B + B_2 - a[i] - b[2n-k-1-i] & i \in [n-k, n-1], & \text{(II)} \\ B_2 - b[2n-k-1-i] - c[i-n] & i \in [n, 2n-k-2], & \text{(III)} \\ B - c[n-k-1] & i = 2n-k-1, & \text{(IV)} \\ B + B_1 - b[3n-k-1-i] - c[i-n] & i \in [2n-k, 2n-1]. & \text{(V)} \end{cases}$$

All formulas form combinations of length bounded by 5 so we can apply the properties (2,3). Observe that the order of each combination equals k , except for $i = 2n-k-1$ where the order is $k+1$. Using the property (3) we reason that $B - c[n-k-1]$ is indeed the maximal forward distance. It remains to show that the minimum lies within the group (I). Note that these are the only combinations that lack $b[n]$. By the property (2) each distance from the group (I) compares less with any other distance because the combinations have the same order (except for the maximal one) and only the latter contains $b[n]$.

We can see that for $k < n$ the condition $M_k < B - B_1$ is equivalent to $c[n-k-1] > \max_{i+j=n-k-1} (a[i] + b[j])$. If there is such a k , i.e., the answer to MAXCONV LOWERBOUND for sequences a, b, c is NO, then $\min_k M_k < B - B_1$ and the return value is less than $\frac{1}{2}(B - B_1)$.

Finally, we need to prove that otherwise $M_k \geq B - B_1$ for all k . We have already acknowledged that for $k < n$. Each matching for $k \geq n$ can be represented as swapping sequences a and c inside

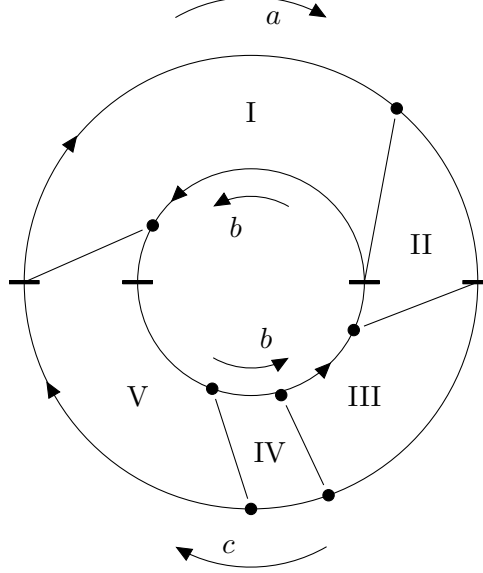


Figure 5: Five areas correspond to the five types of connection between beads. The inner circle represents two repetition of the sequence b . Outer circle consists of the sequence a and then the sequence c .

the necklace x , composed with the index shift by $k - n$. The two halves of the necklace x are analogous so all the prior observations on the matching structure remain valid.

If the answer to MAXCONV LOWERBOUND for sequences a, b, c is YES, then $\forall_{k \in [0, n)} \exists_{i+j=k} a[i] + b[j] \geq c[k]$. The property (1) guarantees that $a \leq c$ so we conclude that $\forall_{k \in [0, n)} \exists_{i+j=k} c[i] + b[j] \geq a[i] + b[j] \geq c[k] \geq a[k]$, and by the same argument as before the cost of the solution is at least $B - B_1$. \square

Observe that both l_∞ -NECKLACE ALIGNMENT and MAXCONV LOWERBOUND admit simple linear nondeterministic algorithms. For MAXCONV LOWERBOUND it is enough to either assign each k a single condition $a[i] + b[k - i] \geq c[k]$ that is satisfied, or guess a k for which none inequality holds. For l_∞ -NECKLACE ALIGNMENT we define a decision version of the problem by asking if there is an alignment of value bounded by K (the problem is self-reducible via binary search). For positive instances the algorithm just guesses k inducing an optimal solution. For negative instances it must hold $M_k > 2K$ for all k . Therefore, it suffices to guess for each k a pair i, j such that $d(x[i], y[k + i \bmod n]) - d(x[j], y[k + j \bmod n]) > 2K$.

We remind that MAXCONV UPPERBOUND (and therefore all other decision problems in the equivalence class) reduces to 3SUM which admits an $\mathcal{O}(n^{1.5} \text{polylog}(n))$ nondeterministic algorithm [14] so in fact there is no obstacle for a subquadratic reduction from MAXCONV LOWERBOUND to MAXCONV UPPERBOUND to exist. However, the nondeterministic algorithm for 3SUM exploits techniques significantly different from ours, including modular arithmetic, and a potential reduction would probably need to rely on some different structural properties of MAXCONV.

7 Conclusions and future work

In this paper we undertake a systematic study of MINCONV as a hardness assumption, and prove subquadratic equivalence of MINCONV with SUPERADDITIVITY TESTING, UNBOUNDED KNAPSACK, 0/1 KNAPSACK, and TREE SPARSITY. An intriguing open problem is to establish the relation between the MINCONV conjecture and SETH.

One consequence of our results is a new lower bound on 0/1 KNAPSACK. It is known that an $\mathcal{O}(t^{1-\epsilon}n^{\mathcal{O}(1)})$ algorithm for 0/1 KNAPSACK contradicts the SETCOVER conjecture [18]. Here, we show that an $\mathcal{O}((n+t)^{2-\epsilon})$ algorithm contradicts the MINCONV conjecture. This does not rule out an $\mathcal{O}(t+n^{\mathcal{O}(1)})$ algorithm, which leads to another interesting open problem.

Finally, it is open whether MAXCONV LOWERBOUND is equivalent to MINCONV, which would imply an equivalence between l_∞ -NECKLACE ALIGNMENT and MINCONV.

Acknowledgements

We would like to thank Amir Abboud, Karl Bringmann and Virginia Vassilevska Williams for helpful discussions.

References

- [1] A. Abboud. personal communication.
- [2] A. Abboud, A. Backurs, and V. V. Williams. Tight hardness results for LCS and other sequence similarity measures. In Guruswami [25], pages 59–78.
- [3] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
- [4] A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In R. A. Servedio and R. Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58. ACM, 2015.
- [5] A. Backurs, P. Indyk, and L. Schmidt. Better approximations for tree sparsity in nearly-linear time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2215–2229. SIAM, 2017.
- [6] I. Baran, E. D. Demaine, and M. Pătraşcu. Subquadratic algorithms for 3sum. In *Proceedings of the 9th International Conference on Algorithms and Data Structures, WADS’05*, pages 409–421, Berlin, Heidelberg, 2005. Springer-Verlag.
- [7] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.
- [8] D. Bremner, T. M. Chan, E. D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, and P. Taslakian. *Necklaces, Convolutions, and $X + Y$* , pages 160–171. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
- [9] K. Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1073–1084. SIAM, 2017.

- [10] K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Guruswami [25], pages 79–97.
- [11] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. On table arrangements, scrabble freaks, and jumbled pattern matching. In P. Boldi and L. Gargano, editors, *Fun with Algorithms, 5th International Conference, FUN 2010, Ischia, Italy, June 2-4, 2010. Proceedings*, volume 6099 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 2010.
- [12] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. On approximate jumbled pattern matching in strings. *Theory Comput. Syst.*, 50(1):35–51, 2012.
- [13] M. R. Bussieck, H. Hassler, G. J. Woeginger, and U. T. Zimmermann. Fast algorithms for the maximum convolution problem. *Oper. Res. Lett.*, 15(3):133–141, 1994.
- [14] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In M. Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270. ACM, 2016.
- [15] C. Cartis and A. Thompson. An exact tree projection algorithm for wavelets. *IEEE Signal Processing Letters*, 20(11):1026–1029, 2013.
- [16] T. M. Chan and M. Lewenstein. Clustered integer 3sum via additive combinatorics. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC ’15, pages 31–40, New York, NY, USA, 2015. ACM.
- [17] F. Cicalese, E. S. Laber, O. Weimann, and R. Yuster. Approximating the maximum consecutive subsums of a sequence. *Theor. Comput. Sci.*, 525:130–137, 2014.
- [18] M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlstrom. On problems as hard as cnf-sat. In *Proceedings of the 2012 IEEE Conference on Computational Complexity (CCC)*, CCC ’12, pages 74–84, Washington, DC, USA, 2012. IEEE Computer Society.
- [19] D. Eppstein. Minimum range balanced cuts via dynamic subset sums. *J. Algorithms*, 23(2):375–385, 1997.
- [20] W. Fenchel. On conjugate convex functions. *Canad. J. Math*, 1(73-77), 1949.
- [21] M. L. Fredman. How good is the information theory bound in sorting? *Theor. Comput. Sci.*, 1(4):355–361, 1976.
- [22] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976.
- [23] A. Gajentaan and M. H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.
- [24] M. Garey and D. Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979.

- [25] V. Guruswami, editor. *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. IEEE Computer Society, 2015.
- [26] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974.
- [27] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
- [28] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
- [29] K. Jansen and S. E. J. Kraft. *A Faster FPTAS for the Unbounded Knapsack Problem*, pages 274–286. Springer International Publishing, Cham, 2016.
- [30] H. Kellerer and U. Pferschy. Improved dynamic programming in connection with an fptas for the knapsack problem. *Journal of Combinatorial Optimization*, 8(1):5–11, 2004.
- [31] K. Koiliaris and C. Xu. A faster pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1062–1072. SIAM, 2017.
- [32] E. S. Laber, W. B. Roncalla, and F. Cicalese. On lower bounds for the maximum consecutive subsums problem and the (min, +)-convolution. In *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014*, pages 1807–1811. IEEE, 2014.
- [33] Y. Lucet. Faster than the fast legendre transform, the linear-time legendre transform. *Numerical Algorithms*, 16(2):171–185, 1997.
- [34] T. M. Moosa and M. S. Rahman. Indexing permutations for binary strings. *Inf. Process. Lett.*, 110(18-19):795–798, 2010.
- [35] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- [36] M. Patrascu. Towards polynomial lower bounds for dynamic problems. In L. J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010.
- [37] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, June 1983.
- [38] G. Toussaint. *The Geometry of Musical Rhythm*, pages 198–212. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
- [39] V. Vassilevska and W. R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *In Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*, 2010.
- [40] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

- [41] R. Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing, STOC '14*, pages 664–673, New York, NY, USA, 2014. ACM.
- [42] V. V. Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In T. Husfeldt and I. A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 17–29. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
- [43] V. V. Williams and R. Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal on Computing*, 42(3):831–854, 2013.
- [44] U. Zwick. All pairs shortest paths in weighted directed graphs-exact and almost exact algorithms. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 310–319. IEEE, 1998.

A The case of non-negative monotonic sequences for SuperAdditivity Testing

Here we will show that in SUPERADDITIVITY TESTING problem we can consider only the case of non-negative monotonic sequence. This is a useful, technical assumption to simplify proofs. Assume that our algorithm for SUPERADDITIVITY TESTING works only for non-negative monotonic sequence. Now we will show how to transform any sequence $(a[i])_{i=0,\dots,n-1}$ to sequence $(a'[i])_{i=0,\dots,n-1}$ that is superadditive iff sequence $a[i]$ is.

First of all note that if $a[0] > 0$, then the sequence is not superadditive for $n > 0$ because $a[0] + a[i] > a[i]$ and we can return **NO**. In the case $a[0] \leq 0$, the 0-th element does not influence the result of the algorithm. So we can put $a'[0] = 0$ to assure non-negativity of a' . Next to guarantee monotonicity choose $C > \max_i \{ |a[i]| \}$. Let

$$a'[i] = \begin{cases} 0, & \text{if } i = 0 \\ Ci + a[i], & \text{otherwise.} \end{cases}$$

Note that sequence $a'[i]$ is strictly increasing and non-negative. Moreover for $i, j > 0$

$$\begin{aligned} a'[i] + a'[j] &\leq a'[i+j] && \iff \\ C \cdot i + a[i] + C \cdot j + a[j] &\leq C(i+j) + a[i+j] && \iff \\ a[i] + a[j] &\leq a[i+j]. \end{aligned}$$

When i or j equals 0 then we have equivalence because $a'[0] = 0$. □

B The Reduction of MaxConv to 0/1 Knapsack

We start with a simple observation, that for UNBOUNDED KNAPSACK (single item can be chosen multiple times) an $\tilde{O}(t^2 + n)$ time algorithm can be obtained by using the standard dynamic programming $\mathcal{O}(nt)$ algorithm.

Theorem 12. *There exists an $\tilde{O}(t^2 + n)$ time algorithm for UNBOUNDED KNAPSACK problem.*

Proof. Our algorithm starts by disregarding all items with weight larger than t . Since we are considering unbounded case, for a given weight we can disregard all items except the one with the highest value, since we can always choose more the most valuable item among the ones of equal weight. We are left with at most t items. So using the standard $\mathcal{O}(nt)$ dynamic programming leads to $\tilde{O}(t^2 + n)$ running time. \square

As we have already shown in Theorem 7 from the perspective of the parameter t it is the best we can hope for, unless n comes into the complexity with exponent higher than 2 or there is a breakthrough for the MAXCONV problem. In this section we complement those results and show that a truly subquadratic algorithm for MAXCONV implies $\tilde{O}(t^{2-\epsilon} + n)$ algorithm for 0/1 KNAPSACK. We will follow Bringmann's [9] near-linear pseudopolynomial time algorithm for SUBSETSUM and adjust it to 0/1 KNAPSACK problem. To do this, we need to introduce some concepts regarding the SUBSETSUM problem from previous works. The key observation is that we can substitute the Fast Fourier Transform in [9] by MAXCONV, and consequently obtain an $\tilde{O}(T(t) + n)$ algorithm⁴ for 0/1 KNAPSACK (where $T(n)$ is the time needed for solving MAXCONV).

B.1 Sum of All Sets for SubsetSum

Let us recall that in the SUBSETSUM problem we are given a set S of n integers together with a target integer t . The goal is to decide whether there exists a subset of S that sums to t .

Horowitz and Sahni [26] introduced the set of *all subset sums* that was later used by Eppstein [19] to solve Dynamic Subset Sum problem. More recently, Koiliaris and Xu [31] used it to show $\tilde{O}(\sigma)$ algorithm for SUBSETSUM (σ denotes the sum of all elements). Later, Bringmann [9] improved this algorithm to $\tilde{O}(n + t)$ (t denotes the target number in SUBSETSUM problem).

The set of *all subset sums* is defined as:

$$\Sigma(S) = \left\{ \sum_{a \in A} a \mid A \subseteq S \right\}.$$

For two sets $A, B \subseteq [0, u]$ the set $A \oplus B = \{x + y \mid x \in A, y \in B\}$ is their join. This join can be computed in time $\mathcal{O}(u \log u)$ by using the Fast Fourier Transform. Namely, we write A and B as polynomials $f_A(x) = \sum_{i \in A} x^i$ and $f_B(x) = \sum_{i \in B} x^i$. Then we can compute the polynomial $g = f_1 \cdot f_2$ in $\mathcal{O}(u \log u)$ time. Polynomial g has nonzero coefficient in front of the term x^i iff $i \in A \oplus B$. At the end we can easily extract $A \oplus B$.

Koiliaris and Xu [31] also noticed that if we want to compute $\Sigma(S)$ for a given S , we can partition S into two sets S_1 and S_2 , recursively compute $\Sigma(S_1)$ and $\Sigma(S_2)$ and join them using FFT. Koiliaris and Xu [31] analysed their algorithm with Lemma 1, which was later used by Bringmann [9].

Lemma 1 ([31], Observation 2.6). *Let g be a positive, superadditive (i.e., $\forall_{x,y} g(x+y) \geq g(x) + g(y)$) function. For a function $f(n, m)$ satisfying*

$$f(n, m) = \max_{m_1 + m_2 = m} \left\{ f\left(\frac{n}{2}, m_1\right) + f\left(\frac{n}{2}, m_2\right) + g(m) \right\}$$

we have that $f(n, m) = \mathcal{O}(g(m) \log n)$.

⁴In the \tilde{O} notation we suppress polylogarithmic factors.

B.2 Sum of all Sets for 0/1 Knapsack

Now we will adapt the notion of sum of all sets to the 0/1 KNAPSACK setting. Here, we use a data structure that for a given capacity stores the value of the best solution we can pack. This data structure can be implemented as an array of size t that keeps the largest value in each cell (for comparison, $\Sigma(S)$ was implemented as a binary vector of size t). To emphasize that we are working with 0/1 KNAPSACK we will use $\Pi(S)$ to denote the array of the values for the set of items S .

If we have two partial solutions $\Pi(A)$ and $\Pi(B)$ we can compute their join denoted as $\Pi(A) \oplus \Pi(B)$. A valid solution in $\Pi(A) \oplus \Pi(B)$ of weight t consists of a solution from $\Pi(A)$ and $\Pi(B)$ that sum up to t (possibly one of them can be 0). Hence $\Pi(A) \oplus \Pi(B)[k] = \max_{0 \leq i \leq k} \{\Pi(A)[k-i] + \Pi(B)[i]\}$. This product is called MAXCONV of array $\Pi(A)$ and $\Pi(B)$. We will use $\Pi(A) \oplus_t^{\max} \Pi(B)$ to denote the MAXCONV of A and B for domain $\{0, \dots, t\}$.

To compute $\Pi(S)$ we can split S into two equal cardinality, disjoint subsets $S = S_1 \cup S_2$, recursively compute $\Pi(S_1)$ and $\Pi(S_2)$ and finally join them in $\mathcal{O}(T(\sigma))$ time (σ is the sum of weights of all items). By Lemma 1 we obtain $\mathcal{O}(T(\sigma) \log \sigma \log n)$ algorithm (recall that naive algorithm for MAXCONV works in $\mathcal{O}(n^2)$ time).

B.3 Retracing Bringmann's Steps

In his algorithm [9] for SUBSETSUM Bringmann uses two key concepts. First, *Layer Splitting* is a very useful observation that an instance (Z, t) can be partitioned into $\log n$ layers $Z_i \subseteq [t/2^i, t/2^{i-1}]$ and $Z_{\lceil \log n \rceil} \subseteq [0, t/n]$. Having such a partition we may infer that for $i > 0$ at most 2^i elements from set Z_i can be used in any solution (otherwise their cumulative sum would be larger than t). Second technique is an application of *Color Coding* [3] leading to fast, randomized algorithm computing all solutions that use at most k elements and their sum is smaller than t . By combining those two techniques Bringmann [9] has shown $\tilde{\mathcal{O}}(t + n)$ time algorithm for SUBSETSUM. Now we will retrace both ideas and use them in 0/1 KNAPSACK context to get an $\tilde{\mathcal{O}}(T(t) + n)$ algorithm. This algorithm improves upon $\tilde{\mathcal{O}}(T(\sigma))$ algorithm from the previous section.

B.3.1 Color Coding

We modify Bringmann's [9] Color Coding technique by using MAXCONV instead of FFT to obtain an algorithm for 0/1 KNAPSACK. We start by randomly partitioning the set of items into k^2 disjoint sets $Z = Z_1 \cup \dots \cup Z_{k^2}$.

Note, that if for every i , the set $|Z_i| \leq 1$, then $\Pi(Z) = \Pi(Z_1) \oplus_t^{\max} \dots \oplus_t^{\max} \Pi(Z_{k^2})$. It is because MAXCONV can choose any subset of Z . For example if $|Z_1| = 2$, then only one item from Z_1 can be chosen to a solution.

Lemma 2. *There exists an algorithm, that computes $\Pi(W)$ in time $\mathcal{O}(T(t)k^2 \log(1/\delta))$, such that for any $Y \subseteq Z$ with $|Y| \leq k$ and weight $i \in [0, t]$ we have $\Pi(Y)[i] \leq \Pi(W)[i]$ with probability $\geq 1 - \delta$ (where $T(n)$ is time needed to compute MAXCONV).*

Proof. We will show that with the split of Z into k^2 partitions, the $\Pi(W) = \Pi(Z_1) \oplus_t^{\max} \dots \oplus_t^{\max} \Pi(Z_{k^2})$ contains solutions at least as good as solutions that use k items (with high probability).

We will use the same argument as in [9]. Assume that the best solution uses $Y \subseteq Z$ items and $|Y| \leq k$. The probability that all items of Y are in different sets of the partition is the same as the

Algorithm 1 COLORCODING(Z, t, k, δ) (cf. [9, Algorithm 1])

```

1: for  $j = 1, \dots, \lceil \log_{4/3}(1/\delta) \rceil$  do
2:   randomly partition  $Z = Z_1 \cup \dots \cup Z_{k^2}$ 
3:    $\Pi(S_j) := \Pi(Z_1) \oplus_t^{\max} \dots \oplus_t^{\max} \Pi(Z_{k^2})$ 
4: end for
5: return  $W$ , where  $W[i] = \max_j \Pi(S_j)[i]$ 

```

probability, that second element of Y is in different set than the first one, the third element is in different set than the first and second item, etc. It is:

$$\frac{k^2 - 1}{k^2} \cdot \frac{k^2 - 2}{k^2} \dots \frac{k^2 - (|Y| - 1)}{k^2} \geq \left(k^2 - \frac{(|Y| - 1)}{k^2} \right)^{|Y|} \geq \left(1 - \frac{1}{k} \right)^k \geq \left(\frac{1}{2} \right)^2 = \frac{1}{4}.$$

So, by repeating $\mathcal{O}(\log(\frac{1}{\delta}))$ times, we get the probability $\geq 1 - \delta$. We need to compute MAXCONV k^2 times. Hence, we get an $\mathcal{O}(T(t)k^2 \log(1/\delta))$ time algorithm. \square

B.3.2 Layer Splitting

We can split our items into $\log n$ layers with weights in $Z_i \subseteq [t/2^i, t/2^{i-1}]$ for $0 < i < \lceil \log n \rceil$ and $Z_{\lceil \log n \rceil} \subseteq [0, t/n]$. With this we can be sure, that only 2^i items from layer i can be chosen to a solution. If we could compute $\Pi(Z_i)$ fast for all i then at the end it suffices to compute their MAXCONV $\mathcal{O}(\log n)$ times. Now we will show how to compute $\Pi(Z_i)$ in $\tilde{\mathcal{O}}(T(t) + n)$ time using Color Coding.

Lemma 3. *There exists an algorithm, that for $Z \subseteq [\frac{t}{2l}, \frac{t}{l}]$ and $\delta \in (0, 1/4]$ computes $\Pi(Z_i, t)$ in time $\mathcal{O}(T(t \log t \log^3(l/\delta)))$, where a single entry of $\Pi(Z_i, t)$ is correct with probability at least $1 - \delta$.*

Algorithm 2 COLORCODINGLAYER(Z, t, ℓ, δ) (cf. [9, Algorithm 3])

```

1: if  $l < \log(l/\delta)$  then return COLORCODING( $Z, t, l, \delta$ )
2:  $m := l / \log(l/\delta)$  rounded up to the next power of 2
3: randomly partition  $Z = Z_1 \cup \dots \cup Z_m$ 
4:  $\gamma := 6 \log(l/\delta)$ 
5: for  $j = 1, \dots, m$  do
6:    $\Pi(S_j) := \text{COLORCODING}(Z_j, 2\gamma t/l, \gamma, \delta/l)$ 
7: end for
8: for  $h = 1, \dots, \log m$  do
9:   for  $j = 1, \dots, m/2^h$  do
10:     $\Pi(S_j) := \Pi(S_{2j-1}) \oplus_{2^h \cdot 2\gamma t/l}^{\max} \Pi(S_{2j})$ 
11:   end for
12: end for
13: return  $\Pi(S_1)$ 

```

Proof. We will use the same arguments as in [9, Lemma 3.2]. First, we split the set Z into m disjoint subsets $Z = Z_1 \cup \dots \cup Z_m$ subsets (where $m = l/\log(l/\delta)$). Then for every partition we compute $\Pi(Z_i)$ using $\mathcal{O}(\log(l/\delta))$ items and probability δ/l using Lemma 2. For every Z_i it takes $\mathcal{O}(T(\log(l)t/l) \log^3(l/\delta))$ time. Hence, for all Z_i we need $\mathcal{O}(T(t) \log^3(l/\delta))$ time, because MINCONV needs at least linear time $T(n) = \Omega(n)$.

At the end, we need to combine sets $\Pi(Z_i)$ in a “binary tree way”. In the first round we compute $\Pi(Z_1) \oplus^{\max} \Pi(Z_2), \Pi(Z_3) \oplus^{\max} \Pi(Z_4), \dots, \Pi(Z_{m-1}) \oplus^{\max} \Pi(Z_m)$. Then at the second round we join the products of the first round in the similar way. We continue until we have joined all subsets. It gives us a significant savings than just computing $\Pi(Z_1) \oplus^{\max} \dots \oplus^{\max} \Pi(Z_m)$ because in round h we need to compute MINCONV up to $\mathcal{O}(2^h t \log(l/\delta)/l)$ place and there are at most $\log m$ rounds. The complexity of joining them is:

$$\sum_{h=1}^{\log m} \frac{m}{2^h} T(2^h \log(l/\delta) t/l) \log t = \mathcal{O}(T(t \log t) \log m).$$

Overall we get roughly (some logarithmic factors could be omitted if we would assume that there exists $\epsilon > 0$, that $T(n) = \mathcal{O}(n^{1+\epsilon})$) that the time complexity of algorithm is $\mathcal{O}(T(t \log t) \log^3(l/\delta))$.

The correctness of the algorithm is based on [9, Claim 3.3]. We take subset of items $Y \subseteq Z$ and let $Y_j = Y \cap Z_j$. The [9, Claim 3.3] says that $\mathbb{P}[|Y_j| \geq 6 \log(l/\delta)] \leq \delta/l$. Thus we can run Color Coding for $k = 6 \log(l/\delta)$ and still guarantee sufficiently high probability of success. \square

Theorem 13 (0/1 KNAPSACK \rightarrow MAXCONV, Theorem 7 restated). *If MAXCONV can be solved in $T(n)$ time then 0/1 KNAPSACK can be solved in time $\mathcal{O}(T(t \log t) \log^3(n/\delta) \log n)$ with probability at least $1 - \delta$.*

Algorithm 3 Knapsack(Z, t, δ) (cf. [9, Algorithm 2])

```

1: split  $Z$  into  $Z_i := Z \cap (t/2^i, t/2^{i-1}]$  for  $i = 1, \dots, \lceil \log n \rceil - 1$ , and  $Z_{\lceil \log n \rceil} := Z \cap [0, t/2^{\lceil \log n \rceil - 1}]$ 
2:  $S = \emptyset$ 
3: for  $i = 1, \dots, \lceil \log n \rceil$  do
4:    $\Pi(S_i) := \text{COLORCODINGLAYER}(Z_i, t, 2^i, \delta/\lceil \log n \rceil)$ 
5:    $\Pi(S) := \Pi(S) \oplus^{\max} \Pi(S_i)$ 
6: end for
7: return  $\Pi(S)$ 

```

Proof. To get 0/1 Knapsack algorithm, as mentioned before we need to split Z into disjoint layers $Z_i = Z \cap [t/2^i, t/2^{i-1}]$ and $Z_{\lceil \log n \rceil} = [0, t/2^{\lceil \log n \rceil - 1}]$. Then compute $\Pi(Z_i)$ for all i and join them using MAXCONV. We present the pseudocode in Algorithm 3. It is based on [9, Algorithm 2]. Overall it takes $\mathcal{O}(T(t \log t) \log n/\delta^3 \log n + T(t) \log n) = \mathcal{O}(T(t \log t) \log^3(n/\delta) \log n)$. \square

Koiliaris and Xu [31] considered the variant of SUBSETSUM where one needs to return if there exists subset that sums up to k for all $k \in [0, t]$. Here, we note that similar extension for 0/1 KNAPSACK is also equivalent to MAXCONV (see Section 2 for a definition of 0/1 KNAPSACK⁺).

Corollary 13.1 (0/1 KNAPSACK⁺ \rightarrow MAXCONV). *If MAXCONV can be solved in $T(n)$ time then 0/1 KNAPSACK⁺ can be solved in $\mathcal{O}(T(t \log t) \log^3(tn/\delta) \log n)$ time with probability at least $1 - \delta$.*

Algorithm 3 returns an array $\Pi(Z)$ where each entry $z \in \Pi(Z)$ is optimal with probability $1 - \delta$. Now if we want to get the optimal solution for all knapsack capacities in $[1, t]$ we need to increase the success probability to $1 - \frac{\delta}{t}$, so that we can use the union bound. Consequently in such a case a single entry is faulty with probability at most δ/t and we can upper bound the event where at least one entry is incorrect by $\delta/t \cdot t = \delta$. In the running time, we get additional $\text{polylog}(t)$ factors.

Finally, for completeness we note that $0/1 \text{ KNAPSACK}^+$ is more general than $0/1 \text{ KNAPSACK}$. $0/1 \text{ KNAPSACK}^+$ returns solution for all capacities $\leq t$. However in $0/1 \text{ KNAPSACK}$ problem we are interested only in capacity equal exactly t .

Corollary 13.2 ($0/1 \text{ KNAPSACK} \rightarrow 0/1 \text{ KNAPSACK}^+$). *If $0/1 \text{ KNAPSACK}^+$ can be solved in $T(t, n)$ then there exists $\mathcal{O}(T(t, n))$ algorithm for $0/1 \text{ KNAPSACK}$.*

At the end we can again check relations of $T(n, t)$ in a ring of reduction. If we follow them from $0/1 \text{ KNAPSACK}$ again to $0/1 \text{ KNAPSACK}$ we will get the following Corollary.

Corollary 13.3. *An $\mathcal{O}((n + t)^{2-\epsilon})$ time algorithm for $0/1 \text{ KNAPSACK}$ implies an $\mathcal{O}(t^{2-\epsilon'} + n)$ time algorithm for $0/1 \text{ KNAPSACK}^+$.*

C 3SUM

Theorem 14 ($\text{MAXCONV UPPERBOUND} \rightarrow 3\text{SUMCONV}$). *If 3SUMCONV can be solved in time $T(n)$ then $\text{MAXCONV UPPERBOUND}$ admits an algorithm with running time $\mathcal{O}(T(n))$.*

The proof heavily utilizes the following lemma. Here $\text{pre}_i(x)$ stands for the binary prefix of x of length i , where the oldest bit is considered the first. In the original statement the prefixed are alternately treated as integers or strings. We modify the notation slightly to work only with integers.

Lemma 4 (Proposition 3.4 [43]). *For three integers x, y, z we have that $x + y > z$, if and only if one of the following holds:*

1. *there exists a k such that $\text{pre}_k(x) + \text{pre}_k(y) = \text{pre}_k(z) + 1$,*
2. *there exists a k such that*

$$\text{pre}_{k+1}(x) = 2 \cdot \text{pre}_k(x) + 1, \tag{2}$$

$$\text{pre}_{k+1}(y) = 2 \cdot \text{pre}_k(y) + 1, \tag{3}$$

$$\text{pre}_{k+1}(z) = 2 \cdot \text{pre}_k(z), \tag{4}$$

$$\text{pre}_k(z) = \text{pre}_k(x) + \text{pre}_k(y). \tag{5}$$

$$\tag{6}$$

Proof of Theorem 14. We are going to translate the inequality $a[i] + b[j] > c[i + j]$ from $\text{MAXCONV UPPERBOUND}$ to an alternative of $2 \log W$ equations. For each $0 \leq k \leq \log W$ we construct two instances of 3SUMCONV related to the conditions in Lemma 4. For the first condition we create sequences $a^k[j] = \text{pre}_k(a[j])$, $b^k[j] = \text{pre}_k(b[j])$, $c^k[j] = \text{pre}_k(c[j]) + 1$. For the second one we choose D to be two times larger than the absolute value of any element and set

$$\begin{aligned}
\tilde{a}^k[j] &= \begin{cases} \text{pre}_k(a[j]) & \text{if } \text{pre}_{k+1}(a[j]) = 2 \cdot \text{pre}_k(a[j]) + 1, \\ -D & \text{otherwise,} \end{cases} \\
\tilde{b}^k[j] &= \begin{cases} \text{pre}_k(b[j]) & \text{if } \text{pre}_{k+1}(b[j]) = 2 \cdot \text{pre}_k(b[j]) + 1, \\ -D & \text{otherwise,} \end{cases} \\
\tilde{c}^k[j] &= \begin{cases} \text{pre}_k(c[j]) & \text{if } \text{pre}_{k+1}(c[j]) = 2 \cdot \text{pre}_k(c[j]), \\ D & \text{otherwise.} \end{cases}
\end{aligned}$$

Observe that if any of the conditions 2 – 4 is not satisfied, then the unrolled formula $\tilde{a}^k[i] + \tilde{b}^k[j] = \tilde{c}^k[i+j]$ contains at least one summand D and thus cannot be satisfied. Otherwise, it reduces to the condition 5.

The inequality $a[i] + b[j] > c[i+j]$ holds for some i, j , if and only if one of the constructed instances of 3SUMCONV returns *true*. As the number of instances is $\mathcal{O}(\log W)$, the claim follows. The 3SUMCONV problem is subquadratically equivalent to 3SUM [36], which establishes a relationship between these two classes of subquadratical equivalence. \square

D Reduction from MaxConv to MaxConv UpperBound

Theorem 15 (MAXCONV \rightarrow MAXCONV UPPERBOUND). *If MAXCONV UPPERBOUND can be solved in time $T(n)$ then MAXCONV admits an algorithm with running time $\mathcal{O}(T(\sqrt{n})n \log n)$.*

Proof. Let us assume that we have an oracle solving the MAXCONV UPPERBOUND, i.e., checking whether $a \oplus^{\max} b \leq c$. First, we argue that by invoking such oracle $\log n$ times we can find an index k for which there exists a pair i, j violating the superadditivity constraint, i.e., satisfying $a[i] + b[j] > c[k]$ where $k = i + j$, if such an index k exists. Let $\text{pre}_k(s)$ be the k -element prefix of a sequence s . The inequality $\text{pre}_k(a) \oplus^{\max} \text{pre}_k(b) \leq \text{pre}_k(c)$ holds only for those k that are less than the smallest value of $i + j$ with a broken constraint. We can employ the binary search to find the smallest k for which the inequality does not hold. This introduces an overhead of factor $\log n$.

Next, we want to show that by using an oracle which finds one violated index, we can in fact find all violated indices. We will take advantage of a technique introduced by Vassilevska and Williams [39]⁵. Let us divide $[0, n-1]$ into $m = \sqrt{n} + \mathcal{O}(1)$ intervals I_0, I_2, \dots, I_m of equal length, except potentially for the last one. For each pair I_x, I_y we can check whether $a[i] + b[j] \leq c[i+j]$ for all $i \in I_x, j \in I_y$ and find a violated constraint (if there exist any) in time $T(\sqrt{n}) \log n$ by translating the indices to $[0, 2n/m] = [0, 2\sqrt{n} + \mathcal{O}(1)]$. After finding a pair i, j that violates the superadditivity we substitute $c[i+j] := K$, where K is a constant exceeding all feasible sums, and continue analyzing the same pair. After no anomalies are detected we move on to the next pair. It is important to note that when an index k violating superadditivity is set to $c[k] := K$, then this value K is also preserved for further calls to the oracle – this way we ensure that each violated index k is reported only once.

For the sake of readability, we present a pseudocode (see Algorithm 4). The subroutine MAXCONVDETECTSINGLE returns the value of $i+j$ for a broken constraint, or -1 if there is no such.

⁵ This technique has been initially used to show a subcubic reduction from $(\min, +)$ -matrix multiplication to detecting a negative triangle in a graph

The notation s^x stands for the subsequence of s in the interval I_x . We assume that elements of c out of a range are equal to K .

Algorithm 4 MAXCONVDetectViolations(a, b, c)

```

1: for  $x = 0, \dots, m - 1$  do
2:   for  $y = 0, \dots, m - 1$  do
3:      $k := 0$ 
4:     while  $k \geq 0$  do
5:        $k := \text{MAXCONVDetectSingle}(a^x, b^y, c^{x+y} \cup c^{x+y+1})$ 
6:       if  $k \geq 0$  then
7:          $c[k] := K$ 
8:          $\text{violated}[k] := \text{true}$ 
9:       end if
10:    end while
11:  end for
12: end for
13: return  $\text{violated}[0, \dots, n - 1]$ 

```

The number of considered pairs of intervals equals $m^2 = \mathcal{O}(n)$. Furthermore, each repetition of the subroutine MAXCONVDetectSingle for a single pair is associated with setting a value of some element of c to K . This can happen only once for each element, so the total number of repetitions is at most n . Therefore, the running time of the procedure MAXCONVDetectViolations is $\mathcal{O}(T(\sqrt{n})n \log n)$.

Having such an algorithm performed, we learn for each $k \in [0, n-1]$ whether $c[k] > \max_{i \in [0, k]} a[i] + b[k - i]$. Hence, we can again use the binary search, this time on each coordinate independently. After running the presented procedure $\log W$ times, the value of $c[k]$ will converge to $\max_{i \in [0, k]} a[i] + b[k - i]$. \square

Corollary 15.1. *If there exists a truly subquadratic algorithm for MAXCONV then it may be assumed to have linear space dependency.*

Proof. The claim follows from the analysis of the Algorithm 4. \square